**Stock Broker Project Report**

**Members of Project: Meddell Bay (#101144294)**

OpenStack information

Username: student

Password: Welcome2MBG!

Public IP address: 134.117.132.7

How-to run-on OpenStack:

* Enter the OpenStack instance
* Go into public/stockMarket directory
* In the terminal, type “node ./express-server.js” to run the server
* There is already a user set within the server if you choose to try logging in
* Username: John
* Password: LOLOLOL2123
* This account on the project already has money within the account
* You can also create a user to access the site

When using the site on the desktop using “ssh -L ####:localhost:3000 [student@134.117.132.7](mailto:student@134.117.132.7)”, the “####” should be set to 3000

Additional information when using the queries:

* To use the “startday” and “enday” queries, the dates are specified as “MM-DD-YYYY ” where January is month 00 (because of JavaScript’s date usage)
* Dates that are not 2 digits (i.e. 6th of December) are to be just the single digit itself
  + - Example: 11-6-2020 = December 6th, 2020

Separate technologies used:

Used UUIDv4 just so that all selling orders can have a specific id and can be accessible specifically to every user and to every stock.

Key Features:

* Event subscription and using a pop-up sidebar to notify the user
* Admin page only accessible through password
* Drop down to view all stocks and a search bar to specify
* Randomized stock prices
* Purchasing, ordering, and selling only at specific times
* Real dates and real time uses (data begins on the 11-6-2020) (does skip days when inactive)

Overall Design:

The overall design of the project is simple but with some more complicated design including how the event handler works for every user. The first design idea that is shown at the beginning of site is that a random person is not able to access any part of the server unless he/she has either created an account on the site or has already been logged in. From being logged in, it then accesses the user page and the basic layout that is in almost all pages.

Every page (not the admin page) all contain the event subscription bar that will pop up from the side of the page whenever one of their subscriptions has been fulfilled and can be notified. The event subscription automatically opens once the get request has been made and then closes after 10 seconds so it does not completely stay on the page forever (can be manually closed if needed by the x button). Every page also includes the title of the site, the time that it is, a dropdown to quickly access a certain stock, a back to user page button, and the logout button. At the end of almost every page is the admin button to access the page

The user page is the standard “/” get request if the person is logged in. This is the case because it makes it easier for the user to get back to their page by either the button at the top right of every page or just by entering the URL and that is it without any parameters or any queries. The page also includes the user’s portfolio, the orders the user has placed, the watchlist, and the event handler. The portfolio contains the necessities of what the user owns and a button to place a sell order. The sell order button only works if the user does not already have a max amount of stocks (owned stocks) in orders already. It is also adjusted whenever someone purchases their stocks. In the orders section, this shows all buying orders and selling orders the user has placed. This includes the cancel button that allows the user to cancel any outstanding orders from their list. But if part of the order has already been processed, the order is final and cannot be undone. The watchlist is a simple table that can be viewed from the user page that contains basic information about the stock at that time. The watchlist is updated whenever the stocks prices have been updated as well. Simple remove button to remove it from the watchlist. The event subscription contains the information of what stock the event is set, the price constraint of when the stock is above or below a certain price (decisions will be in another subsection), an activation button to deactivate or activate the event subscription, and a cancel button.

The admin page is very simple and contains the main functions that are needed to be accessed. Can only be accessed through the button with the specific password, which then gives the user for admin privileges until once said users enters in which upon leaving, the user must re-enter the password through the password. The page also contains the adjustments an admin can do which includes when the stock market is open/closed (based on the hour) as well as to force open the market.

The next page that can be accessed is the index, which is only accessible through the URL using “/stocks” as the parameter and will give out a list of clickable links towards that specific stock. This showcases the current price as well of said stock for all stocks that are requested. The queries that can be used are *maxprice*, *minprice, and symbol.* What “max price” does it find all the stocks that are below the max price that was set, the min price finds all stocks that are above the minimum restraint. The symbol query will find all stocks that contain that specific symbol. From the click of the link, it goes straight to the stocks profile.

The stock page is accessible through the drop down, from the index (“/stocks”) or from the URL themselves. There are 3 queries that can be used, *history, startday, and endday.* The history query is used to specify if it wants to view specific transaction history (true or false) instead of the default of the normal stock history of the days (decision is explained in different subsection). If the history query is not used or is false, the start day and end day will be considered for the normal stock history instead of transaction history. Start day and end day work in a similar manner, but what is entered is in the format of MM-DD-YYYY (month, day, year) where January is considered “00” due to how JavaScript works. The stock page contains the current price of the stock, as well as two buttons which adds to the watchlist and creates an order for that specific user. There are then three tables that showcases the amount of selling orders that can be purchased at the time instead of just ordering, the stock history which includes the highest price, the lowest price, the date, and the amount of transactions that have happened at that day. The transactions table shows the specific transactions of that day which includes who was involved in the sale, the price per stock, and the amount that was bought. The data that is in the stock history and the transaction history is determined by the queries that were used, if non were used, then it gives current day values.

Server-side coding

The server includes all the get and post requests that a user might do, this is for ease of access ass well as everything is accessible all in one script. The beginning involves all the basic information for the server including the stocks, the example user, the user’s array, the admin password and an access of the date functions. On load, the server immediately updates all the prices so new information can be loaded for all the stocks (including the history) and there are no mishaps at the opening of the server. The set interval functions are called at their specific times to update the prices (history) of every stock which then also updates every users watchlist information as well. The do Orders function is called at every interval to then do all the orders that are needed to be fulfilled if the requirements are made. The session contains a secret as well as a cookie that has a max age if there is some sort of idleness. All get requests and post requests are organized in a way, so it is easier to view on the server end. Most of the post requests use “express.json()” so that all strings can be turned into json objects which can be used for editing. The get requests are very simple as they are basic get requests, only ones that are more complicated are the specific stock viewing, and the check events requests. Stock is more complicated just because it would have to check all the queries and only send the specific data (through for loops and try/catch methods) that the user wants. Check events goes through the user’s event subscriptions to find if it is legible to be sent. The post requests become more complicated especially when buying stocks. Buying stocks must edit the user’s portfolio, the account balances, the stocks information (stock history and transaction history for the day), and edit the seller’s information. There are also a lot of if statements to make sure that if anything goes wrong, it is notified.

Decisions that were Made

The hardest decision was to have all the prices randomized and not being able to have it fixed to 2 integers. Randomized prices were due to the fact that exterior trends would be a little more complicated and with randomized prices, everything is faster and updated easily. But with JavaScript, the “toFixed” function would only work with certain float values as it would sometimes turn values into strings. This were multiple ways I tried to fix it, including changing the value in function, having to set temporary values and finding ways to bypass the to fixed function but nothing was able to fix. So prices would be shown at their full length unless it is updated.

The reason that my project uses actual date and time instead of simulating it from day 0 is because I wanted to simulate as much of an actual experience as possible. At the beginning of the semester, I first thought of using actual date because it seemed more interesting and also more realistic. There were some complications of how I wanted to format the dates but after some research about using queries, it is very simple to query when it is MM-DD-YYYY. But since I am not using a database that is not continuous, the only situation is that it would query in the sense that the information is present, so any dates that are not present would skip some dates.

I decided to have just one watchlist per user instead of multiple because the implementation of multiple watchlist would become a lot for the server to have to access and go through every item in every watchlist. Also, this makes it quicker for the user to just view immediately instead of having multiple watchlists that would fill up more of the user’s page. I did attempt to have multiple watchlists but it slowed down the time that the server responds and a user would be able to fill the watchlist to a greater number to slow down the server.

I used a price constraint since I was already in the process of using a price instead of percentages. Also, since all price changes are either added or subtracted only by a small amount (in cents), the percentages that it would change by are significantly smaller. This makes it so the user can also utilize the watchlist to view stocks it wants to buy and use the event subscription as a way of when to sell.

The query for history instead of a “/history” was used because there were issues with getting the client-side JavaScript to the client. I tried using static files and using separate JavaScript files, but nothing seemed to work. Having the query for history was better because I did not have to create any more pug files, JavaScript files, and everything was still viewable to the user.

Buying orders can only be placed once for each user per stock because a user may fill up all the orders for a specific stock and basically buy all the stocks if wanted, this also crashing the server when filled up with too many orders. Instead, a user may be able to order a lot in one order, but it would be fulfilled after multiple iterations of the orders function. Buying orders also goes through every selling order so it can match the order to the closest price as possible instead of the smallest. This again is since a person may order a lot from one specific stock with a high price constraint and take all the lower priced items. It is also so that the orders that have been on the list for a while are also accessible.

What do I like most about the project? Best feature?

What I personally liked about my project was how certain features were simplistic some of the user actions are with the use of prompts as well as notifying a user with alerts when an error may have occurred. It just made purchasing, selling, or subscribing to a stock very easy as it only takes in amounts to submit their action

Another thing that I liked most about my project was the accessibility to each individual stock and the dropdown mechanic. The dropdown was one of my first ideas because I knew that going through a list of stocks on one page was getting a little annoying, so I just made a dropdown instead. A key feature to the dropdown was being able to filter what the user types to show what they want specifically

Additionally, What I like about my project was the use of IRL time and actual dates for the project. Just as in the *decisions that were made section*, I first thought about using actual dates and times to simulate more of an actual stockbroker instead of a simulation where times and prices were updated in a timely basis.

Lastly, my favourite feature from my project is how the event subscription pops-up from the side of the screen when a stock is subscribed too. It does not take up the entire screen nor it does an alert, so the user gets interrupted. It only pops up at the beginning of opening a page and at a time interval. Also, it closes automatically after 10 seconds (or manually) and completely clears the tab. The reason why this feature is my favourite is because of the amount of time that I had to think about the idea of the event subscription, there were ways of just splitting the page but I felt like that destroyed the layout of how the pages looked like.

Testing History:

* Testing direct purchases and how the function works
  + Forgot to edit the seller’s perspective, had to be accessed through find index
  + Editing the stock history was dependent on date
  + There were lots of bugs with why it did not edit amount, forgot to do if statement of when the amount left is 0
  + Had a lot of if statements to make sure everything worked correctly
* Testing the portfolio and placing it in selling
  + Problem for when a user purchases the stock, could not find the selling order
    - Implementation of the UUIDv4 was used to solve the problem
    - Also added the username to every order to make it easier to find who the seller was
  + Problem with not editing the portfolio when someone purchases the stock
    - Forgot to edit in the orders
  + Average price was not calculated correctly
    - Secret values were implemented so that total price paid and the amount purchased (total) were updated in all purchases so the average price is saved easily
* Accessing the admin page
  + User was able to access the page through the URL
    - Made it so that if the certain user does not have the admin Boolean set to true, it would just send that it is not accessible
  + Password was first sent to client which made finding the password to see if the person was able to access
    - Instead, sent the entered password to the server and checked from there
* Testing the queries
  + Trying to use “/history” for transaction history
    - Problematic because the pug file was not able to access the client-side JavaScript.
    - Solved with having history as a query instead of parameter
    - If a stock never had any transactions on that day, it would just do all the history instead